


Parallel Generation of Simple Null Graph Models

Jack Garbus Christopher Brissette George M. Slota

Rensselaer Polytechnic Institute

ParSocial 2020

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

Community Detection and Motif Finding

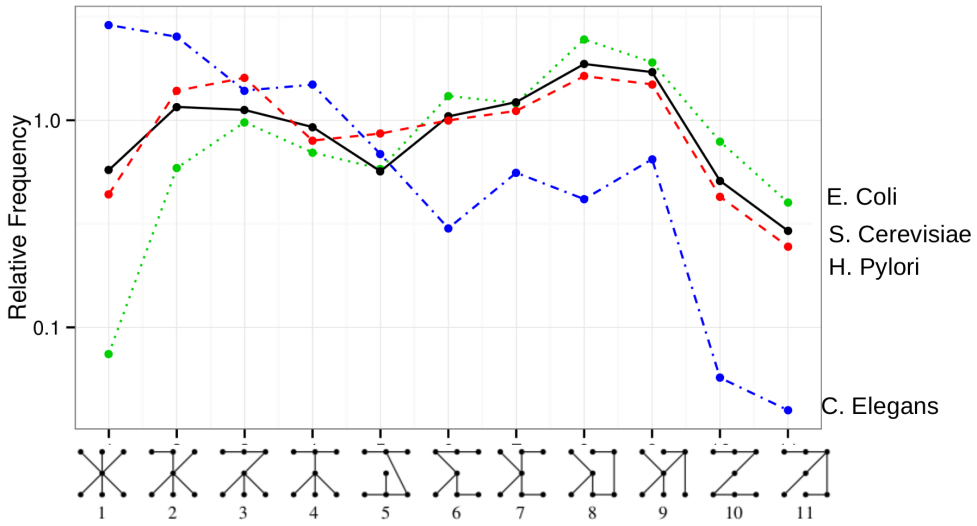
two very widely-used social network analytics

Communities: "Dense" subgraphs in a network.

Most common approach: modularity maximization.

$$\max_{v \in c_v, u \in c_u} Q = \frac{1}{2m} \sum_{(u,v) \in E(G)} \left(A_{u,v} - \frac{\mathbf{k_u k_v}}{\mathbf{2m}}\right) \delta(c_u, c_v)$$

Motifs: "Frequently" occurring subgraphs within a network.



Common Idea: Comparison to a Null Graph Model

Null Graph Model: uniformly random graph, usually matching some property such as a **degree distribution**.

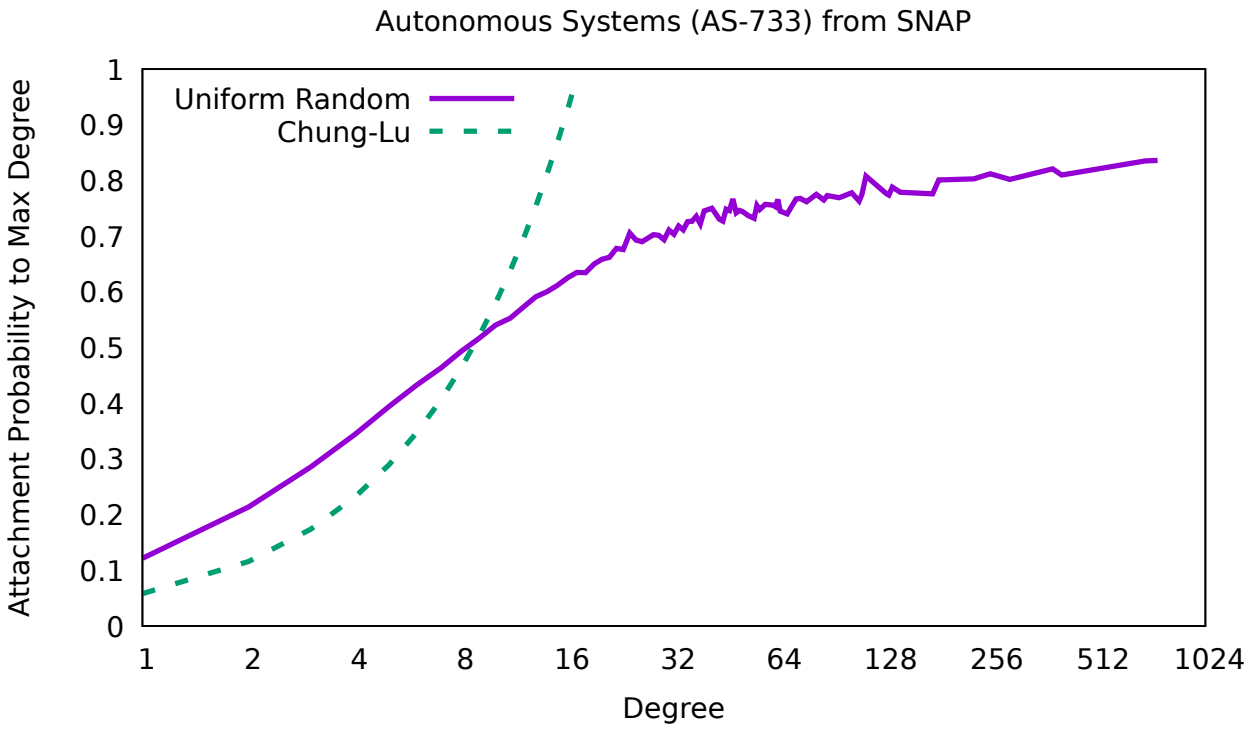
Modularity Maximization: "Dense" subgraphs relative to pairwise attachment probabilities of a *null graph model*.

Motif Finding: "Frequent" subgraphs relative to similar networks or a *null graph model*.

These applications often consider *simple graphs*, but **they commonly utilize an approximation of degree-wise attachment probabilities for non-simple graphs**.

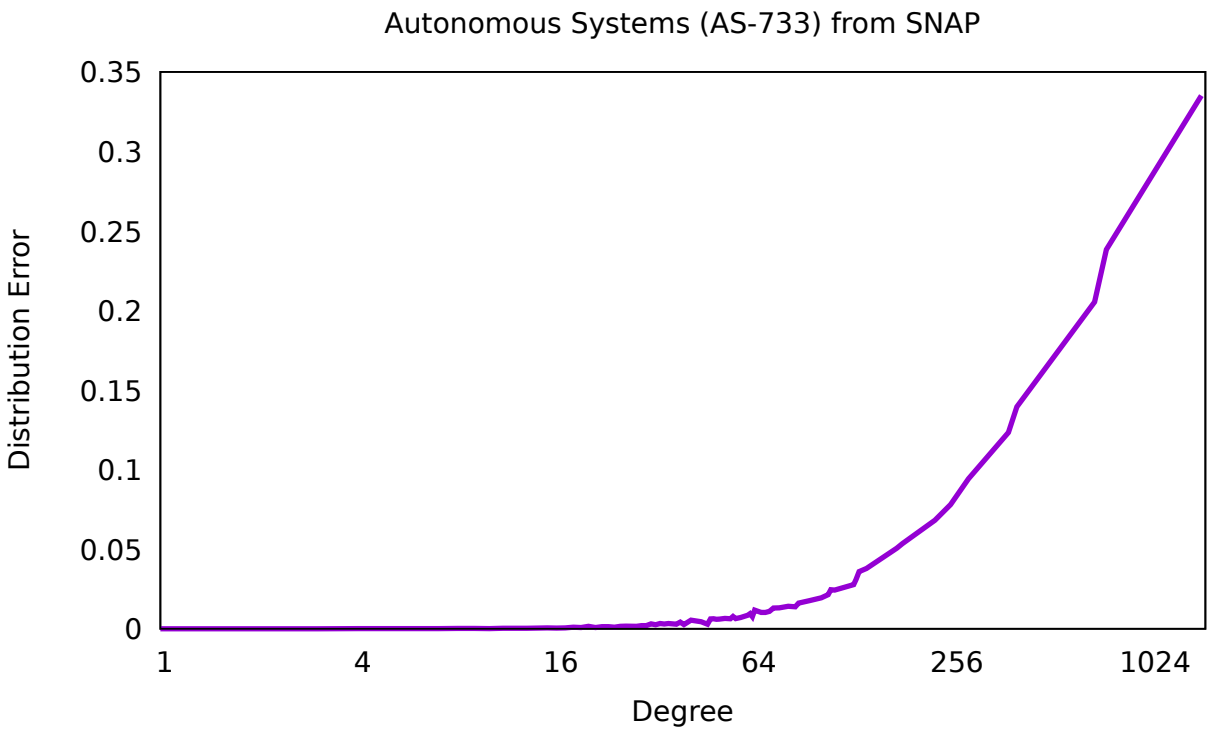
Issue 1: Attachment Probabilities

- $\frac{k_u k_v}{2m}$ is probability of edge between u, v in a uniformly random *loopy multi-graph* (e.g., configuration or Chung-Lu models).
- For skewed and/or dense *simple graphs*, **this approximation is very wrong**. Below: AS-773 empirical vs. approximate.



Issue 2: Graph Generation

- Using $\frac{k_u k_v}{2m}$ to generate simple graphs can result in considerable error in the degree distribution.
- Below: degree distribution error for an *erased configuration model* (multi-edges and loops discarded).



So what does this mean?

Simply put: using $\frac{k_u k_v}{2m}$ to generate graphs or compute network measurements on simple graphs is probably **bad practice**.

- **Modularity Maximization**: attachment probabilities bias towards anti-assortativity in pairwise community membership for large-degree vertices.
- **Motif Finding**: graph generation gives higher assortativity for large-degree vertices and results in combinatorial explosion for subgraph counts.
- See Fosdick et al. 2018 for a study of these considerations and some consequences.

This current work focuses on parallel null graph model generation for motif finding and related applications.

Parallel Gen. of Uniformly Random Simple Graphs

We consider two distinct problems:

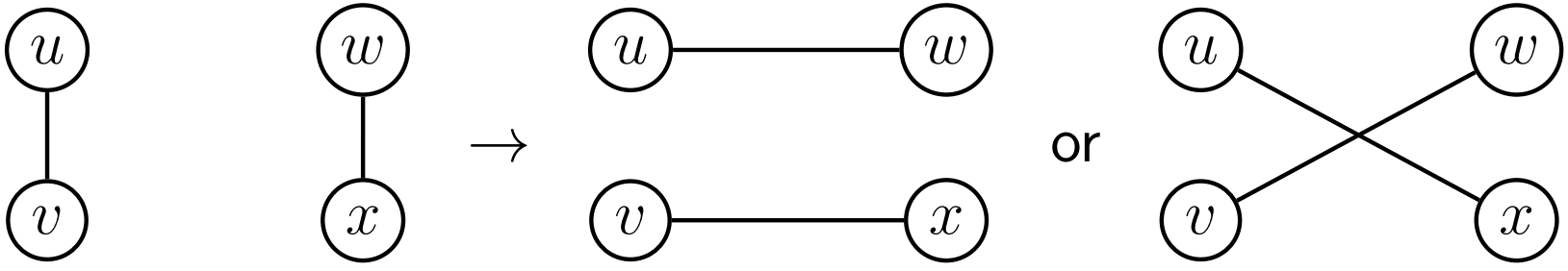
1. Generating a random simple graph matching the degree distribution from a given edge list.
2. Generating a random simple graph matching an input degree distribution.

To address these, we implement the following methods:

- **Method 1**: Implement a scalable procedure for parallel *double-edge swaps* (Problems 1 and 2).
- **Method 2**: Implement a parallel way to generate a simple edge list matching in expectation an input degree sequence by calculating attachment probabilities (Problem 2).

Double Edge Swaps

- Markov Chain Monte Carlo process that retains degree distribution and *eventually* results in graph selected from uniformly random space.
- Randomly select $e = (u, v), f = (w, x)$ and rewire to $e' = (u, x), f' = (v, w)$ or $e' = (u, w), f' = (v, x)$
  – Iterate "some number" of times
- We also need to take care to retain graph simplicity.
  – Discard swaps that will results in multi-edges or loops.



Approach for "Problem 1"

Generating simple random graph from existing edge list

Using an existing edge list, we can sample from the simple graph space by doing an iterative swap procedure on all edges.

```
1:  procedure SwapEdges(E)
2:      for some number of iterations do
3:          T ← ∅                                   ▷ Efficient thread-safe hash Table
4:          for all e = {u, v} ∈ E do in parallel
5:              TestAndSet(T, e)                    ▷ Thread-safe insertion
6:          Permute(E)                              ▷ Parallel Permutation via Shun et al. 2015
7:          for i = 1 ... |E|, i is even do in parallel
8:              e ← E(i), f ← E(i + 1)              ▷ Get swap partners
9:              g, h ← swap(e, f)                   ▷ Perform swap
10:             if TestAndSet(T, g) = false and
11:                 TestAndSet(T, h) = false and
12:                 g and h not self loops then
13:                 E(i), E(i + 1) ← g, h           ▷ Swap successful
14:             else
15:                 E(i), E(i + 1) ← e, f           ▷ Multi-edge or loop identified
16:         clear(T)
```

We validate the resultant graph is uniformly sampled.

Approach for "Problem 2"

Generating from just a degree distribution

For "Problem 2", we need to generate an edge list that matches a degree distribution. From that, we can then perform swaps to get our uniformly selected sample. How do we do it?

- Exact: Havel-hakimi (or similar) generation
  - Runs in $O(m)$ but is not parallelizable
  - We ideally want a $O(\frac{m}{p})$ algorithm
- In Expectation: Bernoulli-style generation
  - Evaluate $\forall u, v \in V$ edge with probability $p_{u,v}$
  - Guarantees simplicity but runs in $O(\frac{n^2}{p})$
  - But, we can use parallel "edge-skipping[1]" to get $O(\frac{m}{p})$
  - **What about the attachment probabilities??**

[1] Slota et al. 2019

Attachment Probabilities

In order to use edge-skipping to output an edge list that matches in expectation an input degree distribution D, we need to find a valid solution for all $P_{i,j}$ attachment probabilities in the following under-determined system:

Chung-Lu probabilities are not a solution to this system.

$$
\begin{aligned}
d_1 = \quad & (n_1 - 1) \times P_{1,1} + n_2 \times P_{1,2} + \ldots = \quad (\sum_{i \in D} n_i \times P_{1,i}) - P_{1,1} \\
d_2 = \quad & n_1 \times P_{2,1} + (n_2 - 1) \times P_{2,2} + \ldots = \quad (\sum_{i \in D} n_i \times P_{2,i}) - P_{2,2} \\
\ldots \quad & \ldots \quad \ldots \\
d_{max} = \quad & n_1 \times P_{d_{max},1} + n_2 \times P_{d_{max},2} + \ldots = \quad (\sum_{i \in D} n_i \times P_{d_{max},i}) - P_{d_{max},d_{max}}
\end{aligned}
$$

$$
P_{i,j} = P_{j,i} < 1 \tag{1}
$$

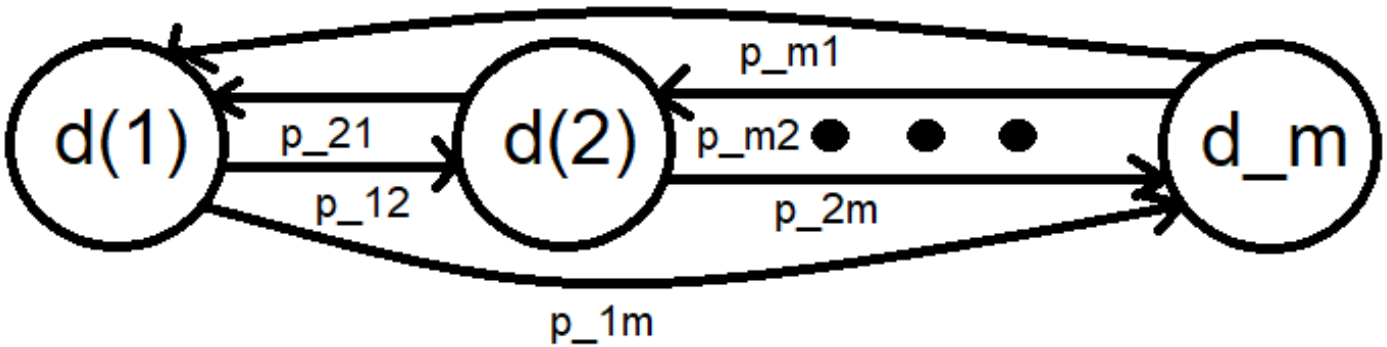
d_i are unique degrees in the degree distribution D, n_i is the number of vertices of degree d_i, and $P_{i,j}$ is the attachment probability between degrees d_i and d_j.

Probability Generation

We use an iterative method for defining edge probabilities:

- Order the degree classes in ascending cardinality.
- Initialize each family with their given number of stubs.
- Start at the first family and connect stubs to each other family depending on how many stubs each family has.
- Decrease the number of stubs for each family by the expected number of connections.

$\rightarrow$ Gives us a parallelizable $O(|D|^2)$ work algorithm



Experimental setup

We compare our methods on the below graphs to:

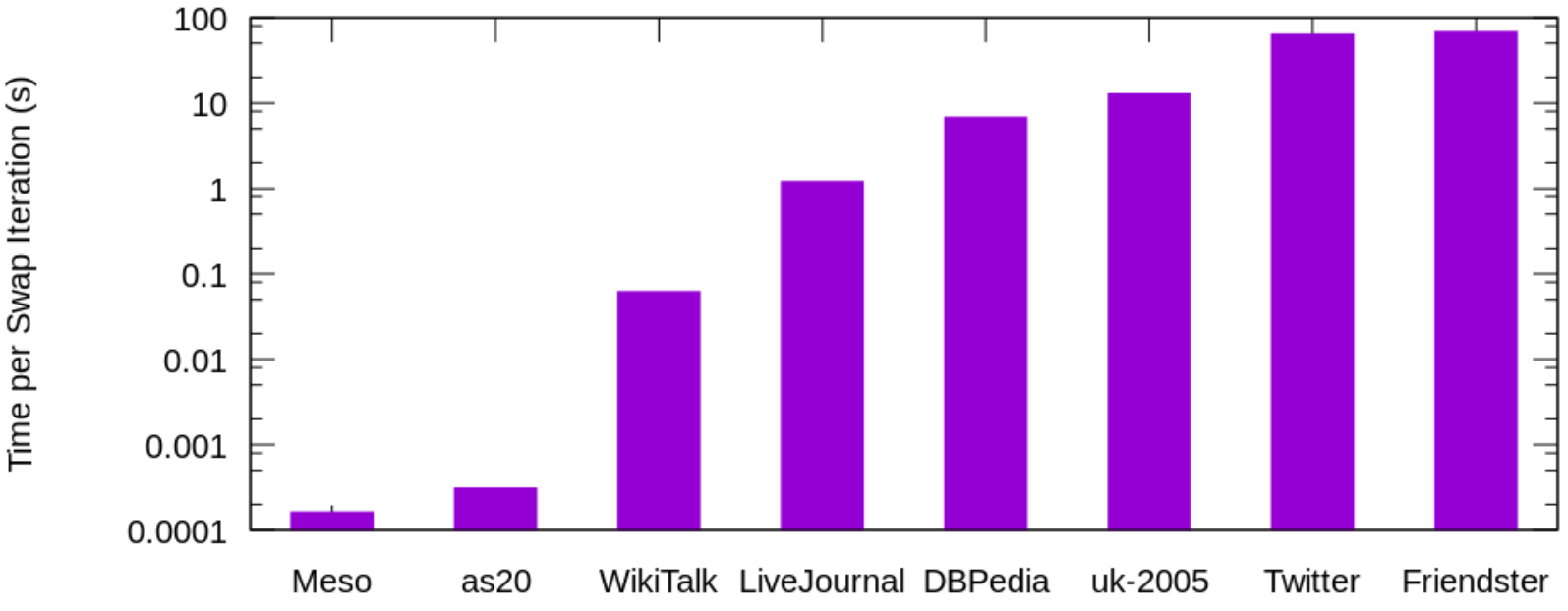
- *Baseline* Chung-Lu model – "$O(m)$"
- *Erased* Chung-Lu model – "$O(m)$ simple"
- *Bernoulli* Chung-Lu model "$O(n^2)$ edgeskip"

Network	n	m	d_{avg}	d_{max}	$\lvert D \rvert$	Source
Meso	1.8 K	3.1 K	3.4	401	31	Shimoda et al.
as20	6.5 K	12.5 K	3.9	1.5 K	83	SNAP
WikiTalk	2.4 M	5.0 M	3.9	100 K	1.8 K	SNAP
DBPedia	67 M	193 M	5.8	7.3 M	4.9 K	Morsey et al.
LiveJournal	4.1 M	27 M	13	2.0 K	945	SNAP
uk-2005	30 M	728 M	49	41 K	5.2 K	LAW
Twitter	39 M	1.4 B	73	56 K	18 K	SNAP
Friendster	40 M	1.8 B	90	5.2 K	3.1 K	SNAP

Method 1: Scalability of edge-swapping

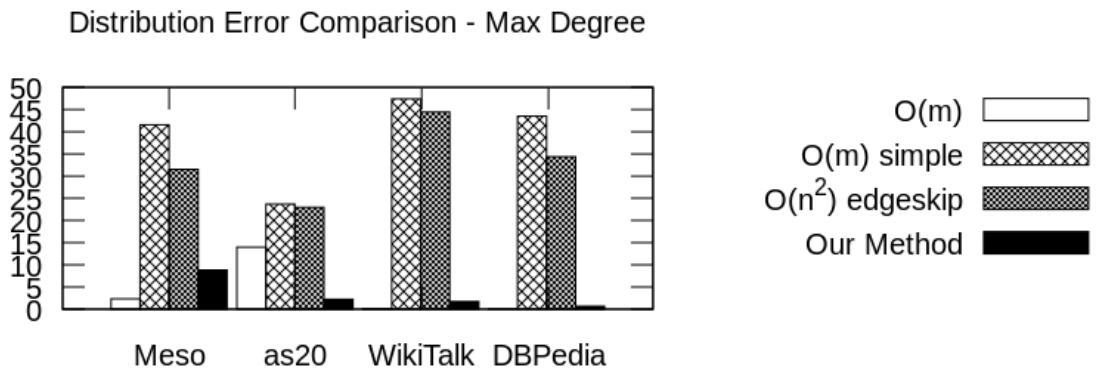
Our edge-swapping routine strong scales very well. Relative to prior work (Bhuiyan et al. 2017), we observe an order-of-magnitude speedup.

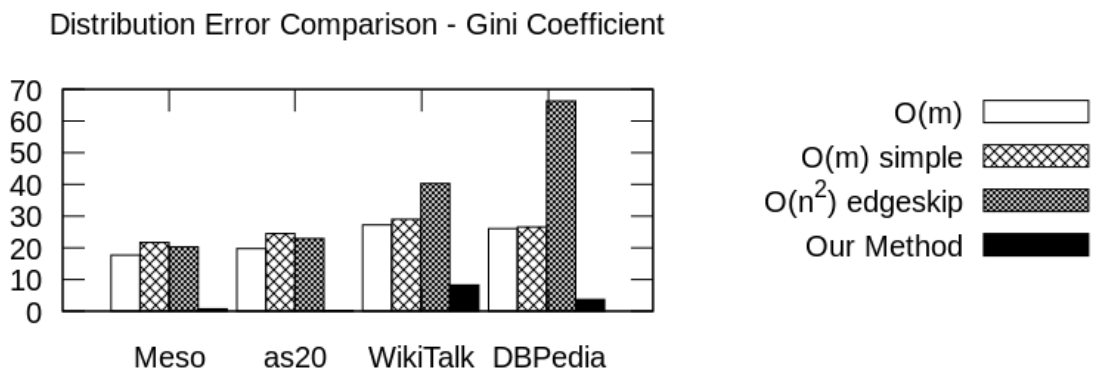
Time for a single iteration (attempting to swap every edge in the edge list) given below for several well-known test inputs:



Method 2: Edge-list generation

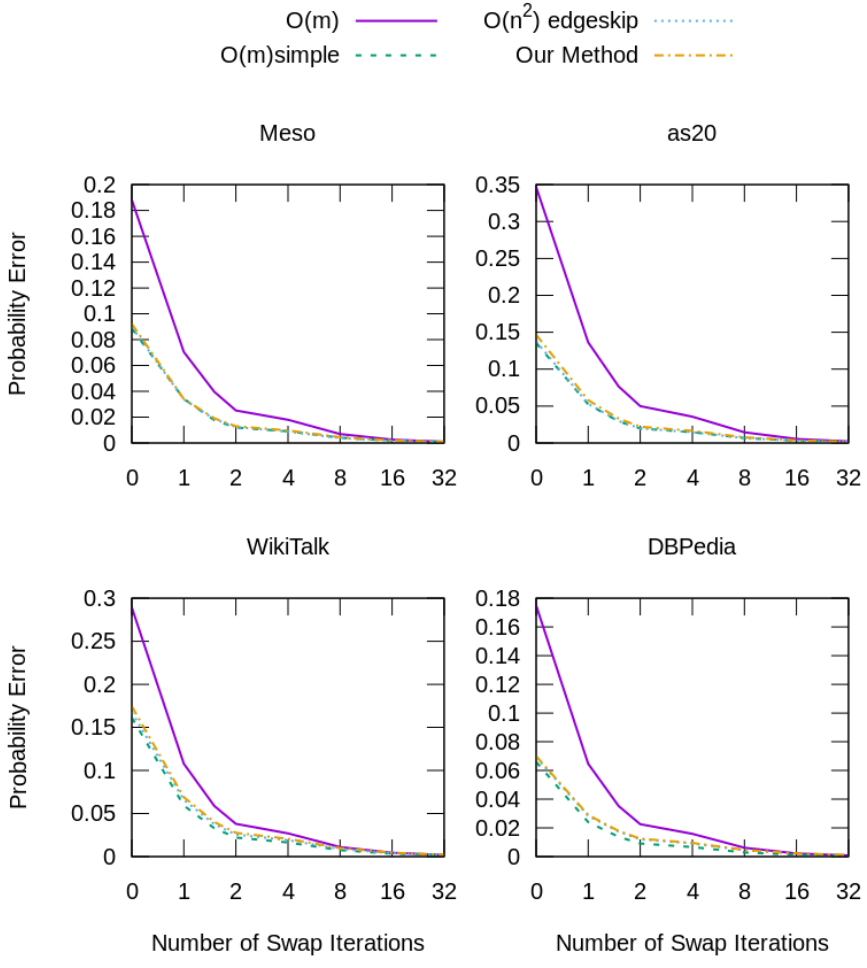
Our method for computing attachment probabilities for edge-skipping outputs an edge list that better matches the input maximum degree (top) and Gini coefficient (bottom).





Convergence towards uniformly random

We compare how many swap iterations it takes for attachment probabilities to converge towards an empirically determined uniformly random graph. We note less than 1% average error after about 3-5 iterations for most generation methods, except for $O(m)$ which is notably slower to converge.



Discussion and open questions

Open questions:

1. **How can we analytically determine uniformly random attachment probabilities for a simple graph?**
   – We have an approximation, but requires $O(n^2|D|^2)$ work and approximation error in practice is unknown.
2. **Can we directly sample from the simple graph space?**
   – Many approaches exist, but are somewhat restrictive (e.g., require $d_{max} < m^{\frac{1}{4}}$).
3. **How many iterations of edge-swapping is required to get a uniformly random graph sample?**
   – Open problem, but we observe about one successful swap per edge is a good approximation.
4. **Other solution methods for our linear system?**
   – Modifying probabilities can impact assortativity, clustering coefficient, etc. of the output edge list.

Conclusions and thanks!

Major takeaways:

- $\frac{k_u k_v}{2m}$ is often "inappropriate" for simple graph generation and applications that use null graph models.
- We develop better faster and better quality methods to quickly output uniformly random graphs, such as for use when comparing motif counts to a null graph model.
- There's still many open questions to consider.

Thank you! Contact below with any questions.

slotag@rpi.edu www.gmslota.com